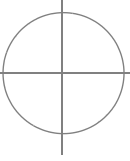


# Einleitung



## E.1 DirectX, C++ und .Net

DirectX kann heute als der Grafikstandard in der Spieleprogrammierung angesehen werden. Praktisch alle modernen Spiele arbeiten mit DirectX, manche (vor allem im Open-Source- und Linux-Umfeld) wahlweise auch mit der Alternative OpenGL. DirectX selbst ist in C/C++ geschrieben und bis vor Kurzem wurden auch Spiele fast ausschließlich in der Sprache C++ programmiert – wobei normalerweise die Spielehersteller eigene *Game-Engines* entwickelt haben, die dem Programmierer Routineaufgaben erleichtern und einen Teil der mit DirectX verbundenen Komplexität vor ihm verbergen. Solche Engines müssen vielfach auch Aufgaben mit übernehmen, die direkt nichts mit Grafik zu tun haben und deshalb von DirectX auch nicht speziell unterstützt werden, wie Datei-Ein- und -Ausgabe, Netzwerkprogrammierung (die Verwendung von DirectPlay wird ja von Microsoft selbst nicht mehr empfohlen), Verarbeitung von XML etc. Die Sprache C++ selbst bietet ja für viele solche Standardaufgaben nur eingeschränkte oder gar keine Unterstützung und die C++-Standardbibliothek STL (Standard Template Library) wird von vielen Programmierern als kompliziert empfunden. Neuere, moderne objektorientierte Programmiersprachen wie Java oder eben auch die .Net-Sprachen C# und Visual Basic.Net haben da mit ihren Klassenbibliotheken weitaus mehr zu bieten und sind auch gerade für Anfänger doch deutlich leichter zu erlernen und zu beherrschen als C++. Aber auch fortgeschrittene Programmierer wissen die Produktivitätssteigerung zu schätzen, die mit der Verwendung solcher Sprachen einhergeht. Es stellt sich daher die Frage, ob man beispielsweise mit C# nicht auch Spiele programmieren kann, und dass die Antwort »Ja« lautet, wissen Sie bereits, sonst hielten Sie dieses Buch nicht in der Hand.

### Managed DirectX

DirectX selbst basiert auf Microsofts COM-Technologie (Component Object Model; eine Technik zum Wiederverwenden von Objekten – »Komponenten« – auf Binärebene). Nun können COM-Objekte mittels so genannter Wrapper-Klassen auch in .Net-Programmen verwendet werden – grundsätzlich spricht also nichts dagegen, DirectX-Programme in C# zu schreiben. Der erste Schritt in diese Richtung war *Managed DirectX* (.Net-Programme werden ja auch als *Managed Code* bezeichnet, da sie der Verwaltung durch die Laufzeitumgebung – der *Common Language Runtime* – unterliegen). Managed DirectX (*MDX*)

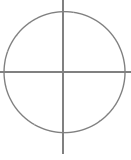
ist ein Wrapper für die wichtigsten DirectX-Funktionen, so dass diese von den .Net-Sprachen verwendet werden können. In Gegensatz zu den DirectX-Funktionen für C++ ist dieser Wrapper zudem durchgängig objektorientiert und Hilfsklassen z.B. für Vektoren und Matrizen mit komfortablen Methoden kapseln die teilweise recht umständlichen DirectX-Funktionsaufrufe.

## XNA

XNA wird allgemein als der Nachfolger von Managed DirectX gesehen, obwohl MDX ebenfalls weiter verfügbar ist. XNA ist ein vollständig neu geschriebenes Framework, wenn auch viele Klassen und Methoden an Managed DirectX erinnern.

## Performance

Im Gegensatz zum C++-Compiler, der echten Maschinencode erzeugt, entsteht beim Kompilieren eines .Net-Programms (so wie auch bei Java-Programmen) ein Zwischencode, der erst zur Laufzeit durch den »Just-In-Time«-(*JIT*)-Compiler wirklich in die Maschinsprache des jeweiligen Prozessors übertragen wird. Zudem bringt die Programmausführung unter der »Aufsicht« der Laufzeitumgebung zwar eine Menge Komfort für den Programmierer (z.B. Garbage Collection), was letztlich die Zuverlässigkeit der Programme erhöht, Robustheit und Sicherheit (welche Programme Zugriff auf welche Systemressourcen haben, kann bei .Net sehr fein eingestellt werden) – letztendlich lässt sich ein gewisser Overhead, der dabei entsteht, jedoch nicht leugnen. Ob solche Sprachen deshalb für die Spieleprogrammierung (wo Performance ja eines der wichtigsten Themen ist) geeignet sind, ist deshalb heiß umstritten und für jede Meinung finden sich auch Tests, die diese untermauern. Am besten probieren Sie es selbst aus: Das DirectX SDK (vgl. Abschnitt 1.2) enthält eine Reihe von Beispielprogrammen sowohl in einer C++- als auch in einer C#-Version (»Managed«). Sie werden feststellen, dass C# bei den meisten dieser Programme nicht oder nur wenig hinter dem entsprechenden C++-Programm zurücksteht. (Dabei spielt allerdings auch geschickte Programmierung eine Rolle: Man *kann* zweifellos ein C#-Programm so schreiben, dass es wesentlich langsamer als ein C++-Programm wird.) Zwar enthält das DXSDK keine Beispiele in XNA, aber die Autorin konnte aus eigener Erfahrung feststellen, dass die Portierung eines Programms von Managed DirectX nach XNA einen weiteren, dramatischen Performance*gewinn* mit sich



brachte – ich führe dies auf die effiziente Programmierung des XNA-Frameworks selbst zurück.

Und wenn man gelegentlich liest, dass Sprachen wie C# ja wohl für Anfänger und einfache Spiele gut genug seien, echte Highend-Anwendungen mit den allermodernsten Effekten aber nach wie vor C++ erfordern, so ist dem zumindest entgegenzuhalten, dass gerade diese allermodernsten Effekte heutzutage ohnehin mit Hilfe von Shadern auf der Grafikkarte berechnet werden und somit gar nichts mehr mit dem Programmcode der eigentlichen Anwendung und der gewählten Programmiersprache zu tun haben. (Shader werden entweder in einer eigenen Assemblersprache oder mittlerweile mehr und mehr in der *High Level Shader Language (HLSL)* geschrieben, vgl. Kap. 9.) Und XNA bietet nicht nur eine ausgezeichnete Unterstützung für die Verwendung von Shadern, sondern zwingt den Programmierer sogar dazu, indem alternative Techniken (die so genannte *Fixed-Function Pipeline*) gar nicht mehr unterstützt werden. Mit XNA setzen Sie also durchaus auf die Zukunft. Leider bringt das auch seine Schattenseiten mit sich: XNA erfordert mindestens eine Grafikkarte, die Shader Model 1.1 unterstützt (2.0 empfohlen) und läuft auch nur unter Windows XP und Windows Vista.

## E.2 Zielgruppe

Um mit diesem Buch sinnvoll arbeiten zu können, sollten Sie schon Programmierkenntnisse haben, am besten natürlich in C#, aber wenn Sie eine objektorientierte Programmiersprache wie C++ oder Java beherrschen, werden Sie sich auch rasch zurechtfinden. Umsteiger von anderen Programmiersprachen haben es etwas schwerer. Wenn Sie noch nie programmiert haben, empfehle ich Ihnen dringend, zunächst zumindest den ersten Teil des E-Books zur C#-Programmierung auf der Begleit-CD durchzuarbeiten, um sich die Grundlagen anzueignen. Andernfalls werden Sie vielleicht die Beispiele des Buchs von der CD kopieren und zum Laufen bringen können, aber sobald Sie eigene Ideen verwirklichen möchten, sind Programmierkenntnisse unverzichtbar, und das Buch ist nicht für absolute Anfänger geschrieben.

Auch Kenntnisse im Umgang mit der Entwicklungsumgebung Visual C# oder Visual Studio allgemein sind hilfreich, notfalls können Sie die Bedienung aber auch »on the fly« lernen.

Über Spieleprogrammierung und 3D-Grafikprogrammierung brauchen Sie noch nichts zu wissen; in den ersten Kapiteln werden die Grundlagen von Anfang an erläutert und wenn Sie die Kapitel der Reihe nach durcharbeiten, werden Sie am Ende so viel Übung haben, dass Sie auch komplexere Projekte im Handumdrehen verwirklichen können.

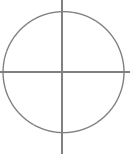
Dabei kann das Buch eine gewisse Zweiteilung nicht leugnen: Die Kapitel 1 bis 6 ermöglichen einen »sanften« Einstieg. Kapitel 7 behandelt die Netzwerkprogrammierung und hat insofern mit Grafik nicht viel zu tun, der zweite Abschnitt des Kapitels erfordert aber einigermaßen gute Kenntnisse in der Sprache C# und dem .Net-Framework. Kapitel 8 behandelt mit den Animationen ein Thema, das an sich technisch komplex ist, allerdings überwiegend anwendungsorientiert vorgestellt wird, so dass auch Anfänger damit zurechtkommen dürften.

Die darauf folgenden Kapitel richten sich aber deutlich an fortgeschrittenere Programmierer und insbesondere die Kapitel 9 und 10, in denen es um Shader geht, erfordern ein gewisses Verständnis für 3D-Grafik und auch eine solide Grundlage in Vektor- und Matrizenmathematik. Bevor Sie sich diesen Kapiteln zuwenden, sollten Sie mindestens den Stoff des Abschnitts 2.9 beherrschen, in dem diese Grundlagen erklärt werden. Andererseits sind dies natürlich gerade für Fortgeschrittene, aber auch für begeisterte Spieler und Fans von Highend-Grafik die interessanten Themen: Bumpmapping, Cubemapping, Darstellung von Wasser, um nur einige Stichworte zu nennen – lassen Sie sich also nicht abschrecken: Die Mühe lohnt sich allemal.

## Was Sie nicht erwartet

In diesem Buch werden keine fertigen Spiele vorgestellt. Sie erlernen alle notwendigen Techniken, um eigene Ideen zu verwirklichen und Ihre Spiele mit faszinierenden visuellen Effekten zu versehen. Teilweise bauen die Beispiele auch aufeinander auf, so dass man schon eine Vorstellung davon bekommt, wie sich das Erlernte zu einem kompletten Spiel zusammenfügen kann.

Die eigentliche Spielhandlung (wenn es nicht nur darum gehen soll, Ihre Mitspieler möglichst effizient niederzuzumetzeln), die Rätsel, die der Spieler lösen, und die Aufgaben, die er oder sie erfüllen muss, die Gags und die Dialoge, die Ökonomie der Spielwelt usw. – kurz all das, was ein gutes Spiel einmalig macht, bleibt aber Ihrer eigenen Fantasie überlassen.



## E.3 Updates

XNA ist eine junge, zukunftsweisende Technologie. Microsoft treibt die Entwicklung mit kurzen Releasezyklen rasch voran – auf der XNA-Creators-Club-Website (<http://creators.xna.com/>) werden regelmäßig neue Beispiele und andere nützliche Informationen veröffentlicht.

Eventuelle Updates für die Beispielprogramme auf der Buch-CD sowie ggf. eine Errataliste finden Sie auf der Webseite des Verlages unter [www.mitp.de/5945](http://www.mitp.de/5945).